CHAPTER 3

# Introducing MongoDB

*"MongoDB is one of the leading NoSQL document store databases. Itenables organizations to handle and gain meaningful insights from Big Data."*

Some leading enterprises and consumer IT companies have leveraged the capabilities of MongoDB in their products and solutions. The MongoDB 3.0 release introduced a pluggable storage engine and the Ops Manager, which has extended the set of applications that are best suited for MongoDB.

MongoDB derives its name from the word "humungous." Like other NoSQL databases, MongoDB also doesn't comply with the RDBMS principles. It doesn't have the concepts of tables, rows, and columns. Also, it doesn't provide features of ACID compliance, JOINS, foreign keys, etc.

MongoDB stores data as Binary JSON documents (also known as BSON). The documents can have different schemas, which means that the schema can change as the application evolves. MongoDB is built for scalability, performance, and high availability.

In this chapter, we will talk a bit about MongoDB's creation and the design decisions. We will look at the key features, components, and architecture of MongoDB in the following chapters.

## History

In the later part of 2007, Dwight Merriman, Eliot Horowitz, and their team decided to develop an online service. The intent of the service was to provide a platform for developing, hosting, and auto-scaling web applications, much in line with products such as the Google App Engine or Microsoft Azure. Soon they realized that no open source database platform suited the requirements of the service.

"We felt like a lot of existing databases didn't really have the 'cloud computing' principles you want them to have: elasticity, scalability, and ... easy administration, but also ease of use for developers and operators," Merriman said. "[MySQL] doesn't have all those properties."[1] So they decided to build a database that would not comply with the RDBMS model.

A year later, the database for the service was ready to use. The service itself was never released but the team decided in 2009 to open source the database as MongoDB. In March of 2010, the release of MongoDB 1.4.0 was considered production-ready. The latest production release is 3.0and it was released in March 2015.MongoDB was built under the sponsorship of 10gen, a New York–based startup.

[1]The Register, Cade Metz, "MongoDB daddy: My baby beats Google BigTable", www.theregister.co.uk/2011/05/25/the_once_and_future_mongodb/), May 25, 2011.


S.G. Edward and N. Sabharwal, *Practical MongoDB*, DOI 10.1007/978-1-4842-0647-8_3

# MongoDB Design Philosophy

In one of his talks, Eliot Horowitz mentioned that MongoDB wasn't designed in a lab and is instead built from the experiences of building large scale, high availability, and robust systems. In this section, we will briefly look at some of the design decisions that led to what MongoDB is today.

## Speed, Scalability, and Agility

The design team's goal when designing MongoDB was to create a database that was fast, massively scalable, and easy to use. To achieve speed and horizontal scalability in a partitioned database, as explained in the CAP theorem, the consistency and transactional support have to be compromised. Thus, per this theorem, MongoDB provides high availability, scalability, and partitioning at the cost of consistency and transactional support. In practical terms, this means that instead of tables and rows, MongoDB uses documents to make it flexible, scalable, and fast.

## Non-Relational Approach

Traditional RDBMS platforms provide scalability using a scale-up approach, which requires a faster server to increase performance. The following issues in RDBMS systems led to why MongoDB and other NoSQL databases were designed the way they are designed:

- In order to scale out, the RDBMS database needs to link the data available in two or more systems in order to report back the result. This is difficult to achieve in RDBMS systems since they are designed to work when all the data is available for computation together. Thus the data has to be available for processing at a single location.
- In case of multiple Active-Active servers, when both are getting updated from multiple sources there is a challenge in determining which update is correct.
- When an application tries to read data from the second server, and the information has been updated on the first server but has yet to be synchronized with the second server, the information returned may be stale.

The MongoDB team decided to take a non-relational approach to solving these problems. As mentioned, MongoDB stores its data in BSON documents where all the related data is placed together, which means everything is in one place. The queries in MongoDB are based on keys in the document, so the documents can be spread across multiple servers. Querying each server means it will check its own set of documents and return the result. This enables linear scalability and improved performance.

MongoDB has a primary-secondary replication where the primary accepts the write requests. If the write performance needs to be improved, then sharding can be used; this splits the data across multiple machines and enables these multiple machines to update different parts of the datasets. Sharding is automatic in MongoDB; as more machines are added, data is distributed automatically.

## JSON-Based Document Store

MongoDB uses a JSON-based (JavaScript Object Notation) document store for the data. JSON/BSON offers a schema-less model, which provides flexibility in terms of database design. Unlike in RDBMSs, changes can be done to the schema seamlessly.

This design also makes for high performance by providing for grouping of relevant data together internally and making it easily searchable.

A JSON document contains the actual data and is comparable to a row in SQL. However, in contrast to RDBMS rows, documents can have dynamic schema. This means documents within a collection can have different fields or structure, or common fields can have different type of data.

A document contains data in form of key-value pairs. Let's understand this with an example:

```
{
"Name": "ABC",
"Phone": ["1111111",
........"222222"
........],
"Fax":..
}
```

As mentioned, keys and values come in pairs. The value of a key in a document can be left blank. In the above example, the document has three keys, namely "Name," "Phone," and "Fax." The "Fax" key has no value.

## Performance vs. Features

In order to make MongoDB high performance and fast, certain features commonly available in RDBMS systems are not available in MongoDB. MongoDB is a document-oriented DBMS where data is stored as documents. It does not support JOINs, and it does not have fully generalized transactions. However, it does provide support for secondary indexes, it enables users to query using query documents, and it provides support for atomic updates at a per document level. It provides a replica set, which is a form of master-slave replication with automated failover, and it has built-in horizontal scaling.

## Running the Database Anywhere

One of the main design decisions was the ability to run the database from anywhere, which means it should be able to run on servers, VMs, or even on the cloud using the pay-for-what-you-use service. The language used for implementing MongoDB is C++, which enables MongoDB to achieve this goal. The 10gen site provides binaries for different OS platforms, enabling MongoDB to run on almost any type of machine.

# SQL Comparison

The following are the ways in which MongoDB is different from SQL.

1. MongoDB uses documents for storing its data, which offer a flexible schema (documents in same collection can have different fields). This enables the users to store nested or multi-value fields such as arrays, hashes, etc. In contrast, RDBMS systems offer a fixed schema where a column's value should have a similar data type. Also, it's not possible to store arrays or nested values in a cell.
2. MongoDB doesn't provide support for JOIN operations, like in SQL. However, it enables the user to store all relevant data together in a single document, avoiding at the periphery the usage of JOINs. It has a workaround to overcome this issue. We will be discussing this in more detail in a later chapter.

3. MongoDB doesn't provide support for transactions in the same way as SQL. However, it guarantees atomicity at the document level. Also, it uses an isolation operator to isolate write operations that affect multiple documents, but it does not provide "all-or-nothing" atomicity for multi-document write operations.

# Summary

In this chapter, you got to know MongoDB, its history, and brief details on design of the MongoDB system. In the next chapters, you will learn more about MongoDB's data model.